Virtual Memory:

Definition of Page:

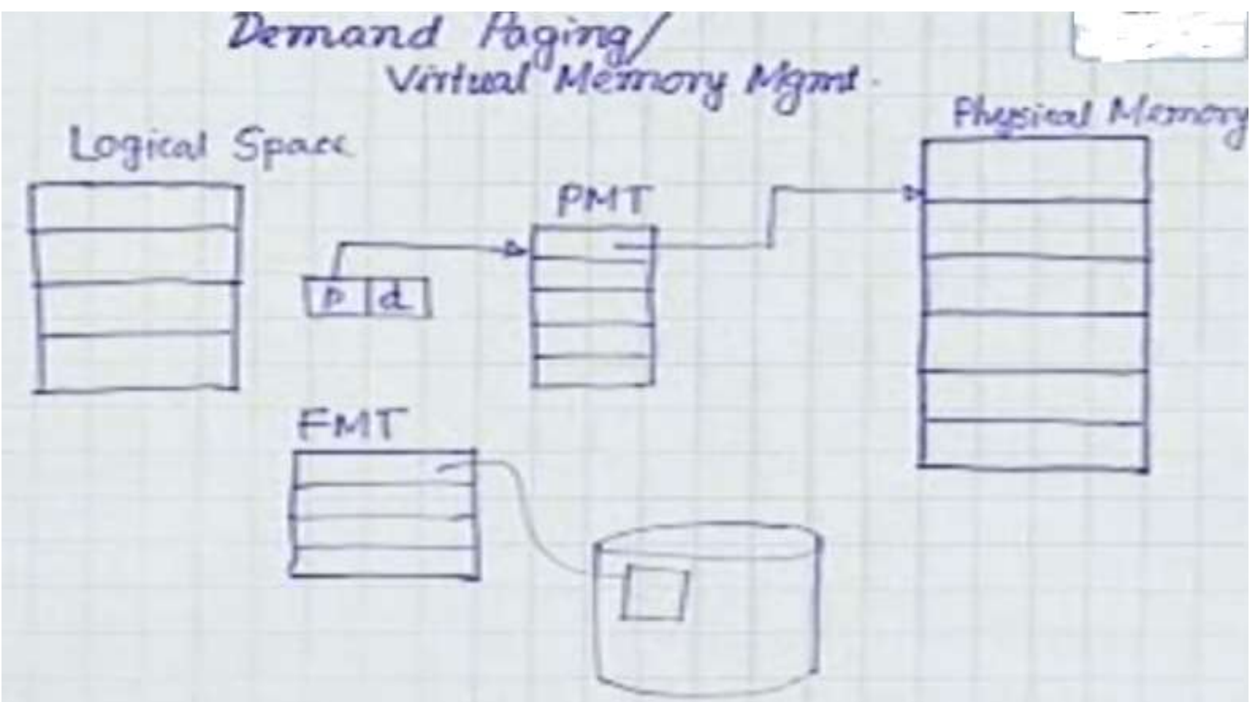
In computer operating systems, paging is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the **operating system** retrieves data from secondary storage in same-size blocks called **pages**.

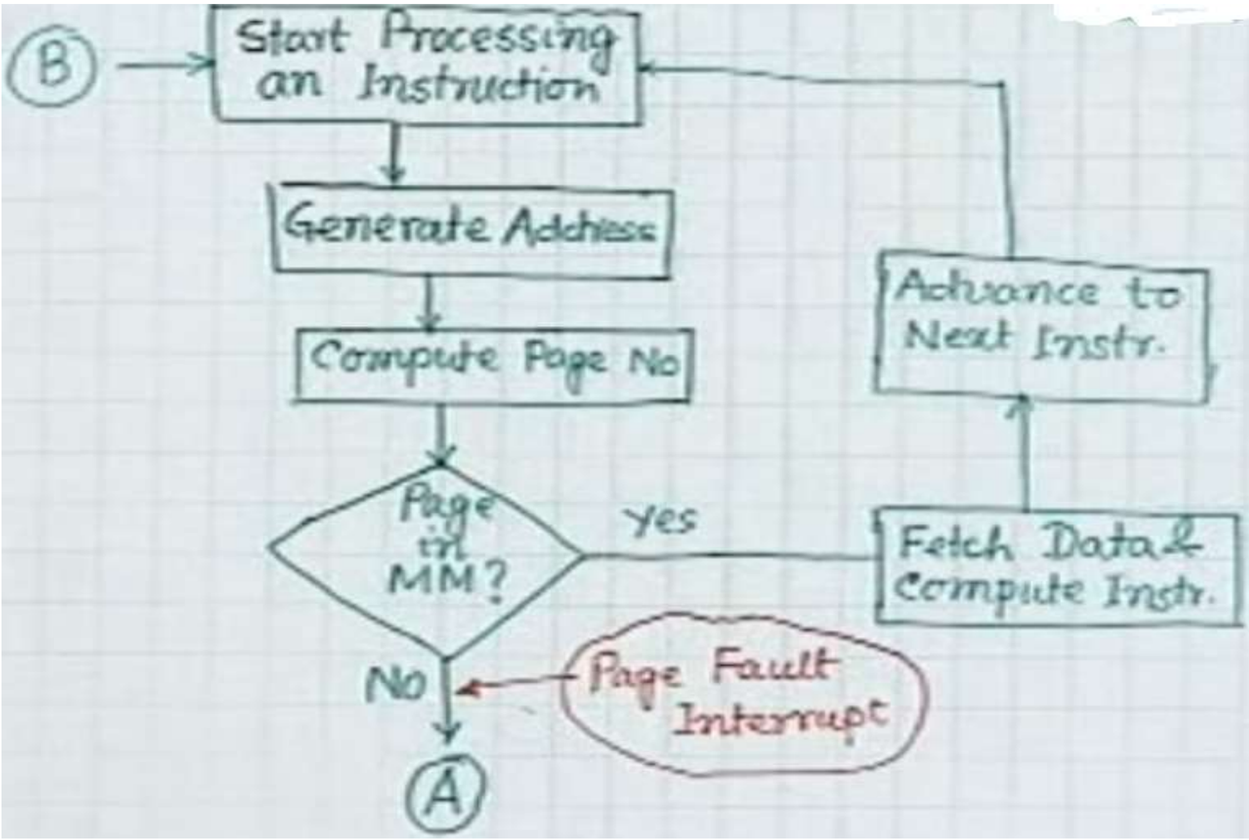
Short version: "page" means "virtual page" (i.e. a chunk of virtual address space) and "page frame" means "physical page" (i.e. a chunk of physical memory).

The page frame is the storage unit (typically 4KB in size) whereas the page is the contents that you would store in the storage unit i.e the page frame. For e.g the RAM is divided into fixed size blocks called page frames which is typically 4KB in size, and each page frame can store 4KB of data i.e the page.

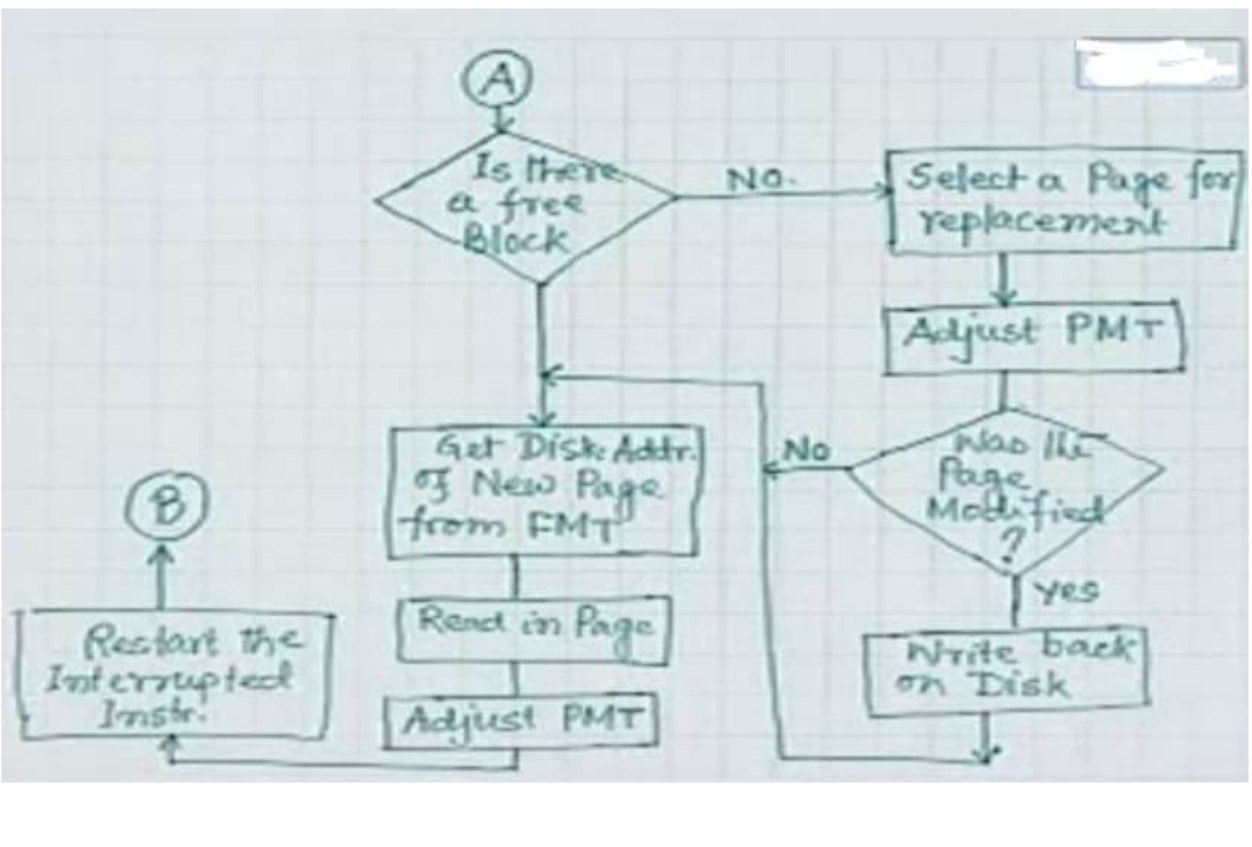
Page is what you want to store and page frame is where you want to store.Page - logical addresses (addresses referred in a code / program)page frame - physical addresses (addresses present in RAM / primary memory).

When using paging, the main memory is partitioned into equal fixed-size chunks that are relatively small, and each process is also divided into small fixed-size chunks of the same size. Then, the chunks of a process, known as pages, are assigned to available chunks of memory, known as frames or page frames.

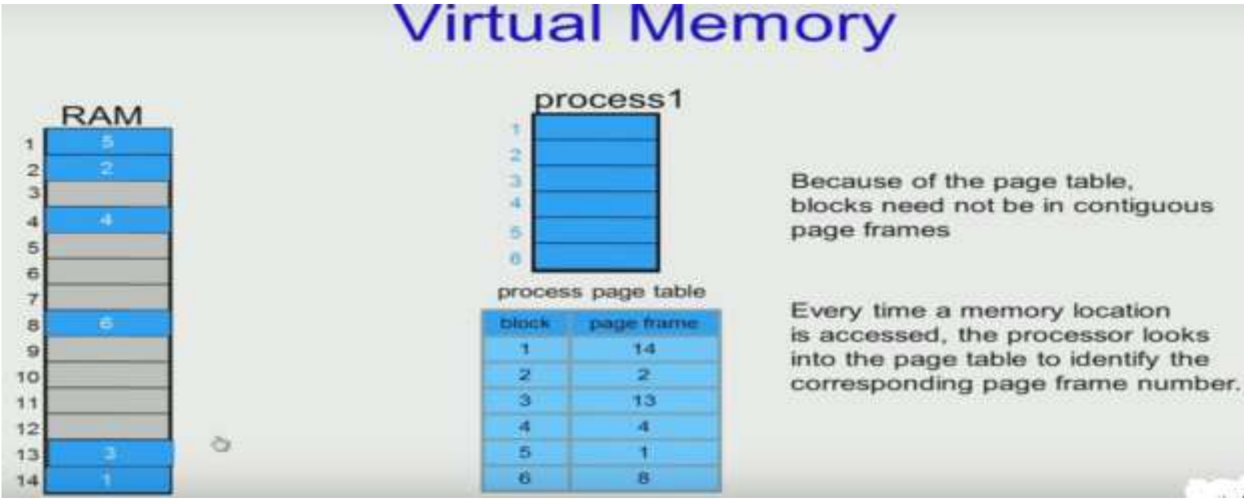


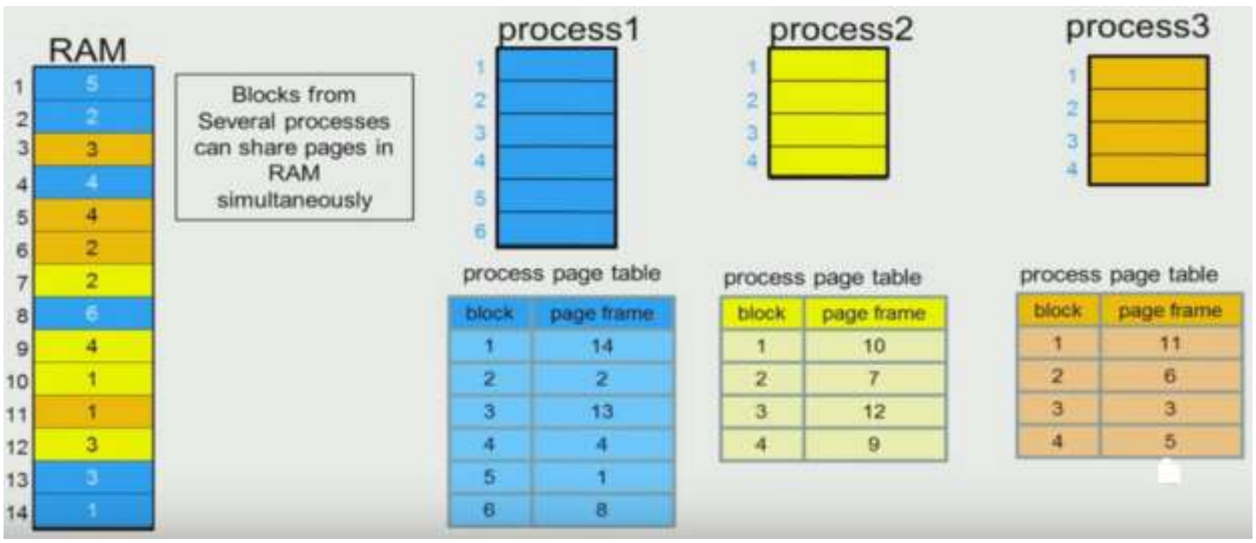


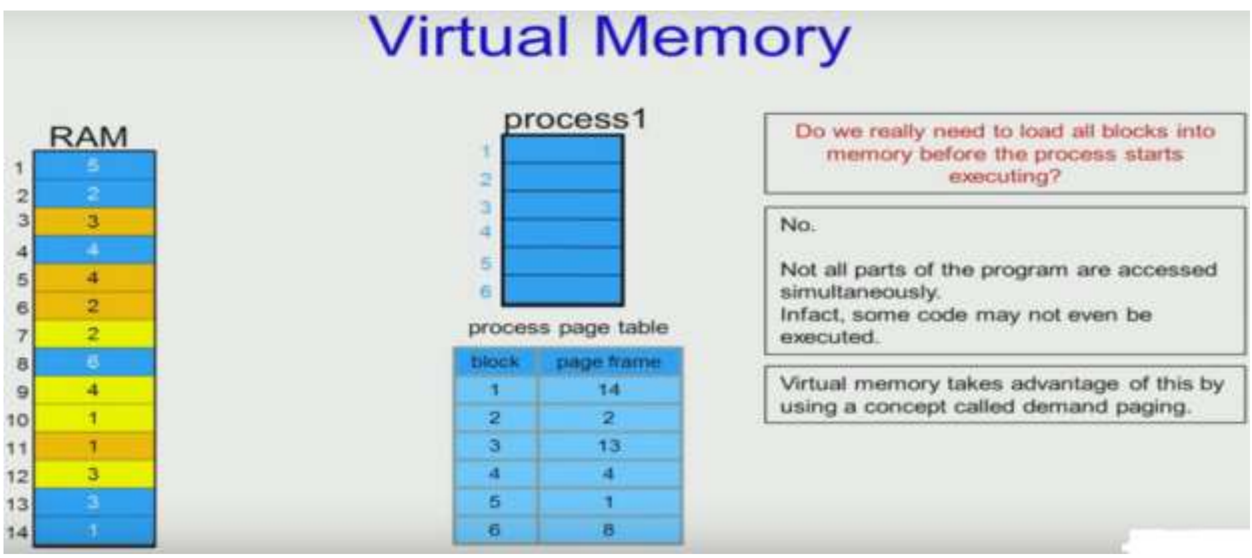
B
Start Processing an Instruction
Generate Address
Compute Page No
Page in MM?
yes
No
Fetch Data & Compute Instr.
Advance to Next Instr.
Page Fault Interrupt
A



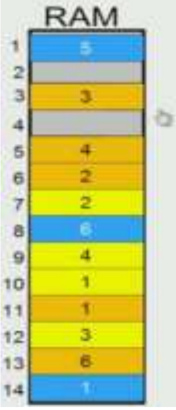
A
Is there a free Block
No.
Select a Page for replacement
Adjust PMT
Was the Page Modified ?
No
Yes
Write back on Disk
Get Disk Addr. of New Page from FMT
Read in Page
Adjust PMT
Restart the Interrupted Instr.
B

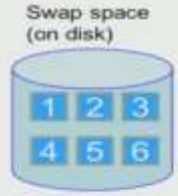

Virtual Memory
RAM
process1
Because of the page table, blocks need not be in contiguous page frames
Every time a memory location is accessed, the processor looks into the page table to identify the corresponding page frame number.
process page table
block | page frame
1 | 14
2 | 2
3 | 13
4 | 4
5 | 1
6 | 8


RAM
Blocks from Several processes can share pages in RAM simultaneously
process1
process2
process3
process page table
block | page frame
1 | 14
2 | 2
3 | 13
4 | 4
5 | 1
6 | 8
process page table
block | page frame
1 | 10
2 | 7
3 | 12
4 | 9
process page table
block | page frame
1 | 11
2 | 6
3 | 3
4 | 5


Virtual Memory
RAM
process1
process page table
block | page frame
1 | 14
2 | 2
3 | 13
4 | 4
5 | 1
6 | 8
Do we really need to load all blocks into memory before the process starts executing?
No.
Not all parts of the program are accessed simultaneously.
Infact, some code may not even be executed.
Virtual memory takes advantage of this by using a concept called demand paging.

Demand Paging





process page table in RAM

block	page frame	P
1	14	1
2		0
3		0
4		0
5	1	1
6	8	1

P ← present bit

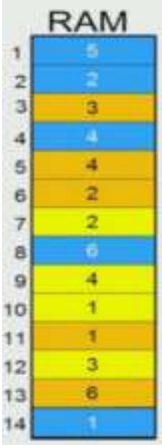
Pages are loaded from disk to RAM, only when needed.

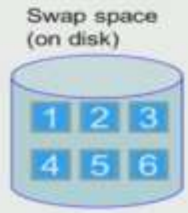
A 'present bit' in the page table indicates if the block is in RAM or not.

If (present bit = 1){ block in RAM}
else {block not in RAM}

If a page is accessed that is not present in RAM, the processor issues a page fault interrupt, triggering the OS to load the page into RAM and mark the present bit to 1

Demand Paging





process page table in RAM

block	page frame	P
1	14	1
2	2	1
3		0
4	4	1
5	1	1
6	8	1

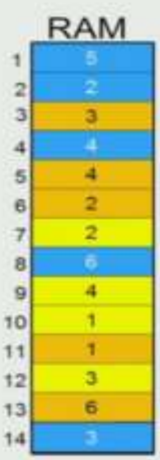
If there are no pages free for a new block to be loaded, the OS makes a decision to remove another block from RAM.

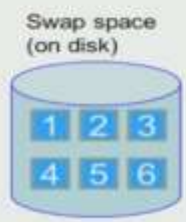
This is based on a replacement policy, implemented in the OS.

Some replacement policies are
- First in first out
- Least recently used
- Least frequently used

The replaced block **may** need to be written back to the swap (swap out)

Demand Paging





process page table in RAM

block	page frame	P
1	14	0
2	2	1
3	14	1
4	4	1
5	1	1
6	8	1

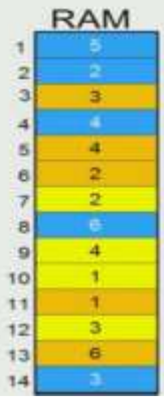
If there are no pages free for a new block to be loaded, the OS makes a decision to remove another block from RAM.

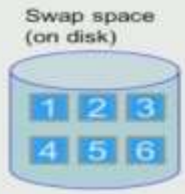
This is based on a replacement policy, implemented in the OS.

Some replacement policies are
- First in first out
- Least recently used
- Least frequently used

The replaced block **may** need to be written back to the swap (swap out)

Demand Paging





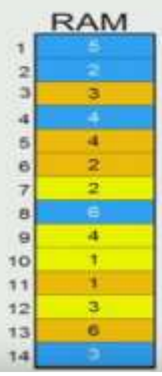
process page table in RAM

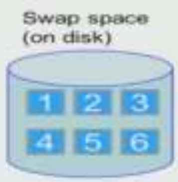
block	page frame	P	D
1	14	0	1
2	2	1	1
3	14	1	0
4	4	1	1
5	1	1	0
6	8	1	1

The **dirty bit**, in the page table indicates if a page needs to be written back to disk

If the dirty bit is 1, indicates the page needs to be written back to disk.

Demand Paging





Protection bits, in the page table determine if the page is executable, readonly, and accessible by a user process.

process page table in RAM ← protection bits

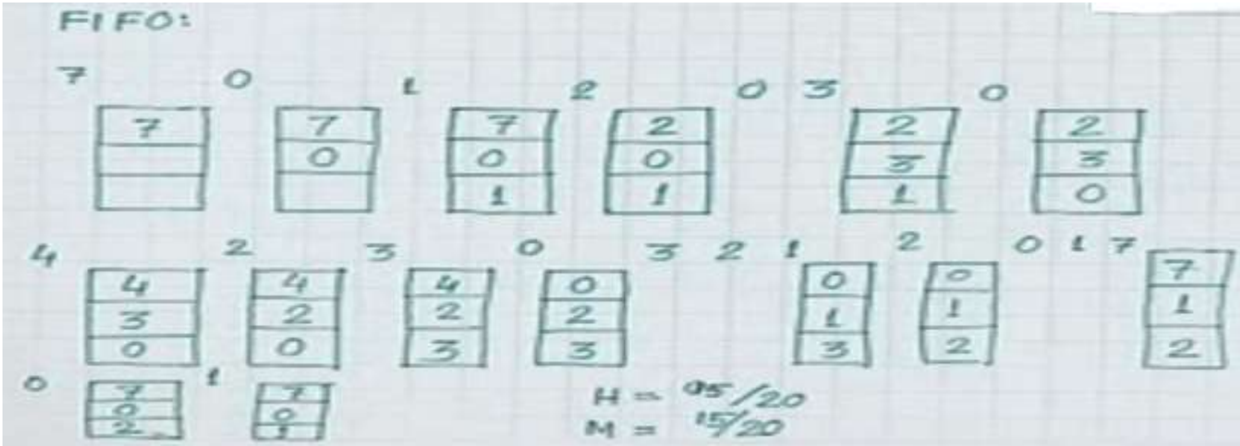
block	page frame	P	D	
1	14	0	1	11
2	2	1	1	10
3	14	1	0	00
4	4	1	1	11
5	1	1	0	01
6	8	1	1	10

FIFO: First-in- First- out.

OPTIMAL:

LRU: Least Recently Used.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7,
0, 1.



Belady's Anomaly Problem : Generally page faults are reduced by increasing the page frame, but sometimes page faults are increasing though the page frame is increased.This problem is called Belady's anomaly.

Belady's Anomaly

- Also called FIFO anomaly.
- FIFO page replacement algorithm :

  "Replace a page that was bought into memory first"
- The page fault rate should decrease as we increase the number of frames.

- For few page reference string, as we increase the number of frames the page fault rate also increases instead of decreasing.
- Commonly experienced in FIFO page replacement algorithm.
- Demonstrated by Laszlo Belady in 1969.
- As page frames increases the page fault also increases.
- Till 1969, it was believed that an increase in the number of page frames would result in fewer page faults.

Example:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5			5	5	
	2	2	2	1	1	1			3	3	
		3	3	3	2	2			2	4	
*	*	*	*	*	*	*			*	*	

No. of Page Faults = 9

1	1	1	1			5	5	5	5	4	4
	2	2	2			2	1	1	1	1	5
		3	3			3	3	2	2	2	2
			4			4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

No. of Page Faults= 10

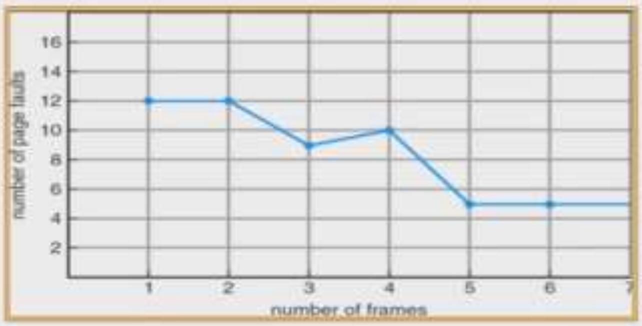
3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	0	0	0	4			4	4	
	2	2	2	3	3	3			1	1	
		1	1	1	2	2			2	0	
*	*	*	*	*	*	*			*	*	

No. of Page Faults = 9

3	3	3	3			4	4	4	4	0	0
	2	2	2			2	3	3	3	3	4
		1	1			1	1	2	2	2	2
			0			0	0	0	1	1	1
*	*	*	*			*	*	*	*	*	*

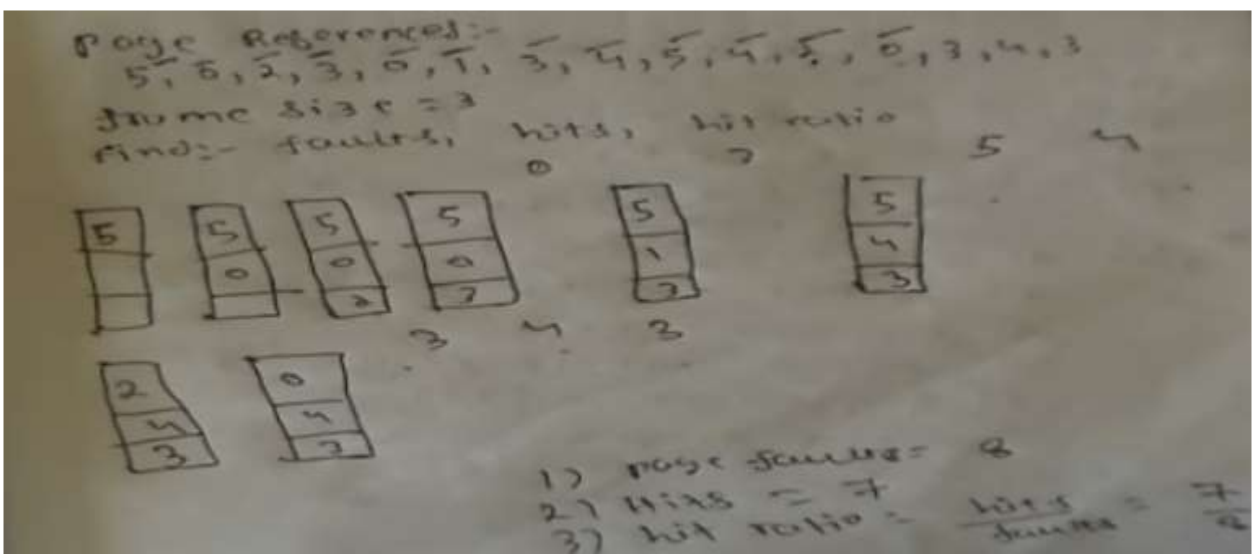
No. of Page Faults= 10

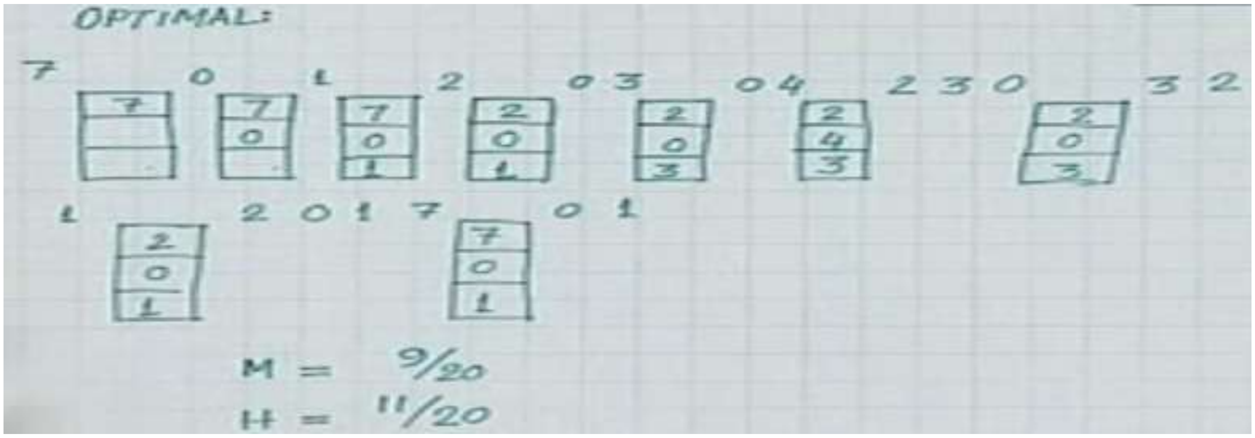
- Figure below shows the curve of page faults versus the number of frames available.



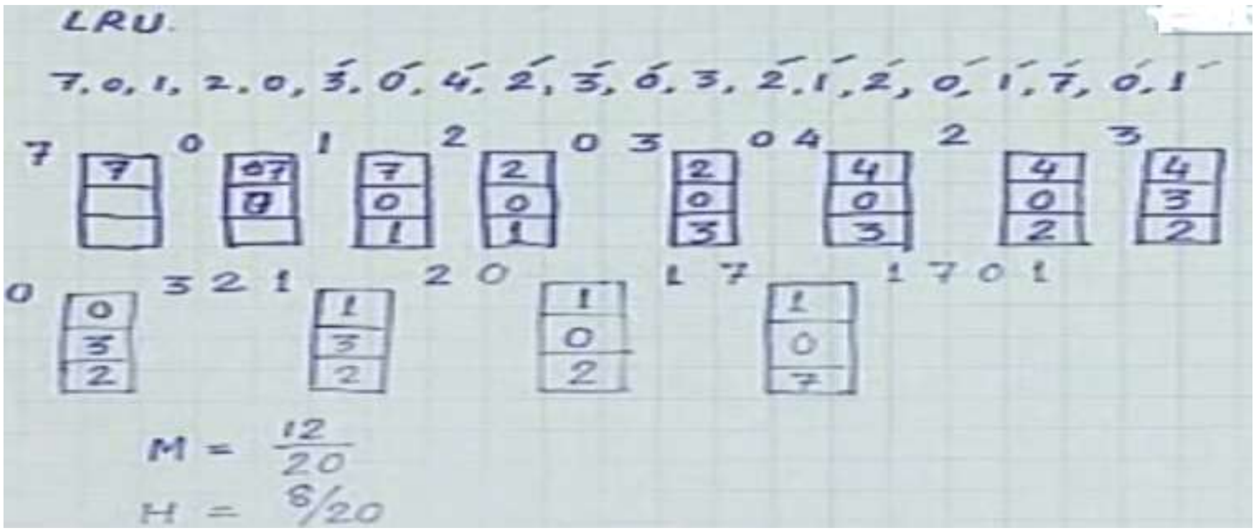
- The number of faults for four frames is greater than the number of faults for three frames.

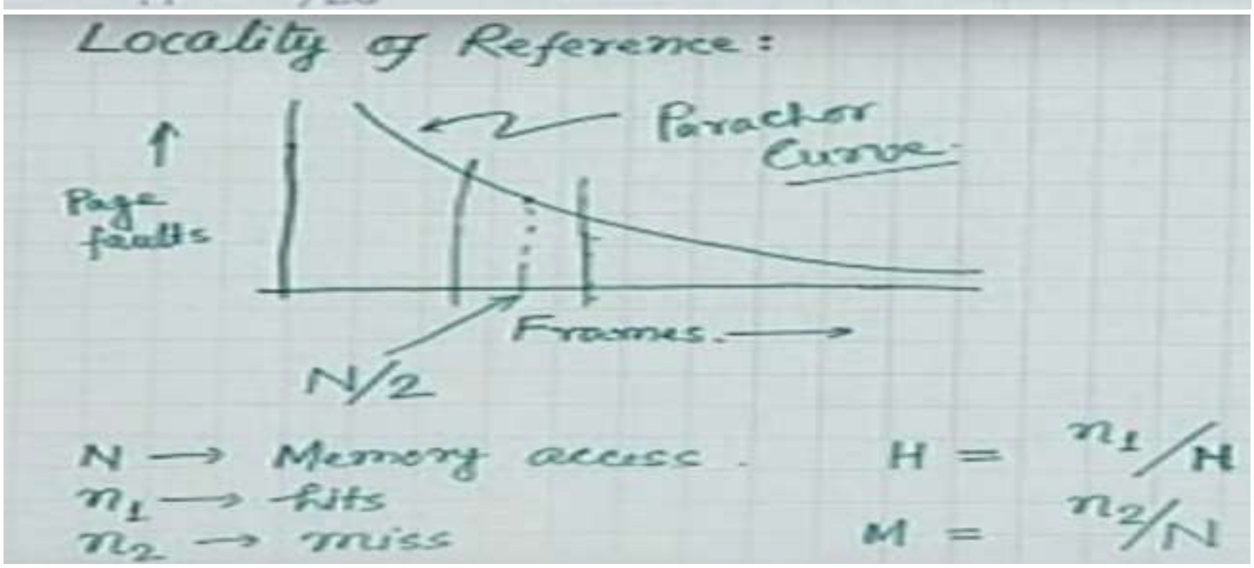
Optimal page replacement algorithm: Which page will be used after longest time period that will replace by the new one.





Least Recently Used (LRU) Algorithm: Which page is used least recently that will be replaced by new page.





Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **Least Frequently Used (LFU) Algorithm**:
    - Replaces page with the smallest reference count
    - Intuition is that active pages have high reference counts
- **Most Frequently Used (MFU) Algorithm**:

- Replaces the page with the highest reference count
- Intuition is that pages with small reference counts were probably just paged into memory recently and has yet to be used in the active working set

- Both approaches are somewhat uncommon and do not approximate OPT very well.

Least Frequently used Algorithm:

Which page is used less number time is replaced by the new page. When page fault occurs the inserted page frequency will increase and replaced page frequency will decrease.